

TOPPERS/ASPを利用した RISC-V HiFive1 boardの評価

2019年12月13日
キオクシア株式会社
メモリ技術研究所
システム技術研究開発センター
山田 真大

本日のアジェンダ

1. 背景
2. 目的
3. アプローチ
4. 作業過程で発生した問題
5. 問題に対する対処
6. 計測結果
7. まとめ

RISC-Vの盛り上がり

- RISC-V Foundation
- ISA（命令セットアーキテクチャ）がOpenであること
 - 実装がOpenである とは言っていないが、Openとなっているものが多数ある
- スクラップアンドビルドにより本当に使用する命令セットだけを採用
 - シンプル＝面積が小さくなる、反面コードサイズは増える傾向
 - 標準拡張命令の取捨選択が可能
- ツールチェーン、シミュレータ、OS移植などエコシステムへの取り組み
- 拡張命令
 - アプリケーションのパフォーマンス追求
 - 賛否あり：移植性の問題、メンテナンスコスト

RISC-VとLinux®の共通点と相違点：RISC-VはLinuxと似ているか？

構成の取捨選択ができること

- RISC-V: 命令セットの選択
- Linux: Linux kernel moduleの選択

Foundationの存在

- RISC-V Foundation
- The Linux Foundation

Open Source?(似ているようで似ていない)

- RISC-Vは、大学、企業等が各自で実装をopenにしている。実装に使用する言語も異なる
- Linux kernelはrepositoryが1つ

⇒実装が分散していると、開発リソースが分散してOSSとしてメリットは低い

組込み向けCPUとして評価を実施

組込み機器に使用できるCPUの選択肢が増えるのはよい

⇒RISC-Vが既存の組込み向けCPUの代替になるか？そのために評価したい

⇒CPU性能は各方面でベンチマーク評価されている

組込み機器での利用を想定してリアルタイム性/RTOSの観点で評価

- **Code Size(text size)**
- **perfによるジッタ計測**

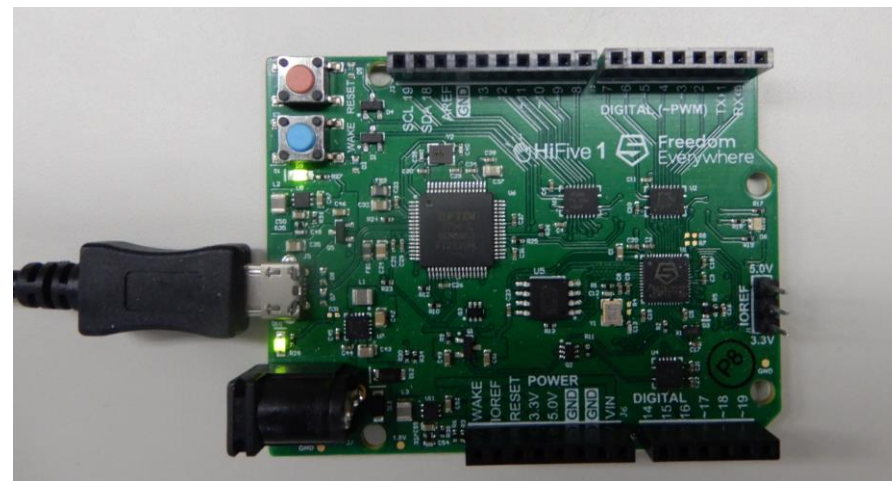
HW面の評価は、今回含まず

- ゲート数
- 周波数の限界性能の見積もり

TOPPERS/ASP on HiFive1でcode sizeとperfの計測

評価に使用する構成

- HW : SiFive社のHiFive1(not rev. B)
 - 入手が容易 (だった)
 - FPGA版もあり実装がオープン
- RTOS : TOPPERS/ASP
 - アーキテクチャ非依存部分の実装によりコードサイズの比較ができる
 - 性能測定用のperfがあらかじめ用意されている



参考値として、NUCLEO-F401RE (Arm® Cortex®-M4) も計測する。ただし、CoreMark性能 : 3.42 CoreMarks/MHz(※1)を見る限り、性能比較するのは不適切。

※1: <https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4>

HiFive1 boardとFE310-G000について

- Arduino互換RISC-Vボード
- FE310-G000 RISC-V SoC搭載
 - RV32IMAC : RV32標準命令 (I) , (M) ultiply, (A) tomic, (C) ompressed
 - 命令キャッシュ : 16KB
 - DTIM : 16KB
 - 性能
 - データシート : 2.73 CoreMarks/MHz (※1)
 - 実測 : 1.496 CoreMarks/MHz (※2)
- SPI Flash memory

※1: https://sifive.cdn.prismic.io/sifive%2Ffeb6f967-ff96-418f-9af4-a7f3b7fd1dfc_fe310-g000-ds.pdf

※2 : <http://msyksphinz.hatenablog.com/entry/2017/03/22/014143>

TOPPERS/ASPの紹介

- TOPPERS/ASPについて (※1)
 - ITRON仕様をベースとして、完成度の高いリアルタイムカーネルを実現したもの
 - 主な適用対象は高い信頼性・安全性・リアルタイム性を要求される組込みシステム
 - ソフトウェア規模の面では、プログラムサイズ（バイナリコード）が数十KB～1MB程度のシステムを主な適用対象。
 - ASP（Advanced Standard Profile）= μ ITRON4.0仕様のスタンダードプロファイル準拠のTOPPERS/JSPカーネルを拡張・改良する形で開発したもの
- 他にもメモリ保護(HRPカーネル)やマルチコア（SMPカーネル）などがある
- ASP3：ASPカーネルにtickless。コンフィギュレータがRubyで実装など。
- 2019年5月にRISC-V（HiFive1ボード）対応版ASPを公開

※ 1 : <https://www.toppers.jp/asp-kernel.html>

TOPPERS/ASPのビルド環境を構築してビルド⇒コードサイズ計測（１）

TOPPERS/ASPのRISC-V向けビルド環境の構築（Debian10）

必要な手続き	内容
toolchainのインストール	SiFive社HPよりダウンロード：riscv64-unknown-elf-gcc-8.3.0-2019.08.0-x86_64-linux-ubuntu14.tar.gz
aspソースコード・ツールの取得	TOPPERS HPよりダウンロード 共通部：asp-1.9.3.tar.gz ターゲット依存部：asp_arch_riscv32_gcc-1.9.3.tar.gz コンフィギュレータ：cfg-linux-static-1_9_3.gz
ホスト環境のmulti arch設定（コンフィギュレータが32bit binaryのため）	\$ sudo dpkg --add-architecture i386 \$ sudo aptitude update \$ sudo aptitude install -y libc6-dev-i386
ソースコードの一部修正	コンパイラ名変更：riscv32-unknown-elf→riscv64-unknown-elf STACK_SIZEの変更：デフォルト4K > 1536 バイト以下に

TOPPERS/ASPのビルド環境を構築してビルド⇒コードサイズ計測（２）

コードサイズ計測用にsampleをビルド

```
$ mkdir asp/OBJ-SAMPLE  
$ cd asp/OBJ-SAMPLE  
$ ../configure -T ../target/hifive1_gcc/  
$ make clean && make depend && make
```

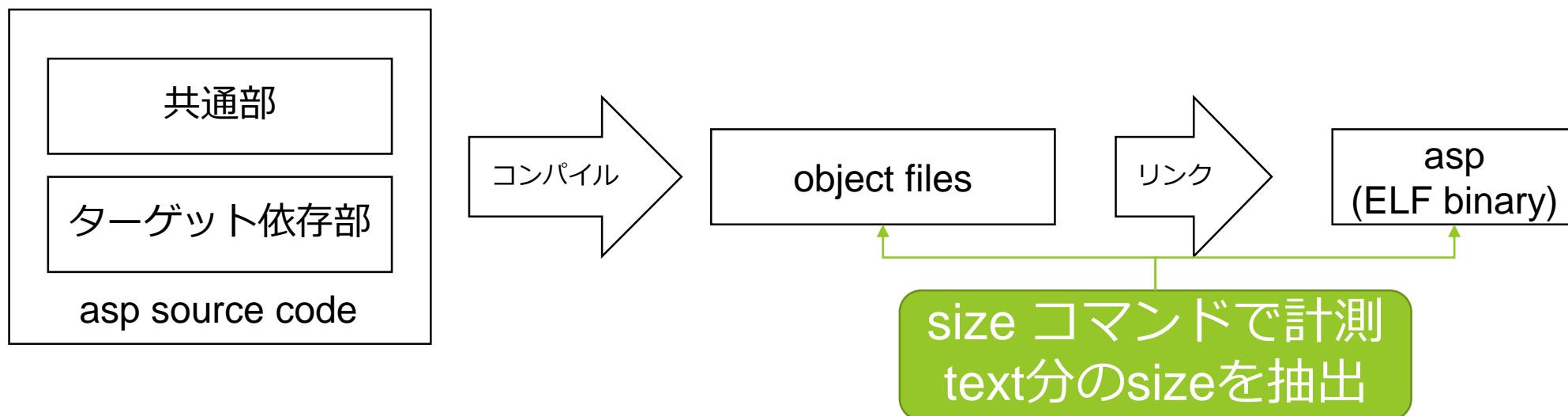
遅延時間計測用にperfのビルド（例：perf0）

```
$ mkdir asp/OBJ-PERF0  
$ cd asp/OBJ-PERF0  
$ ../configure -T ../target/hifive1_gcc/ -A perf0 -a ../test -U "test_lib.o histogram.o"  
$ make clean && make depend && make
```

TOPPERS/ASPのビルド環境を構築してビルド⇒コードサイズ計測（3）

sampleのビルド結果（*.oやasp）に対してsizeコマンドで計測

- 共通部の各object fileの計測
 - ターゲット依存部やコンフィギュレータによって生成される部分は除去
- asp（ELFバイナリ）の計測（include ターゲット依存部）



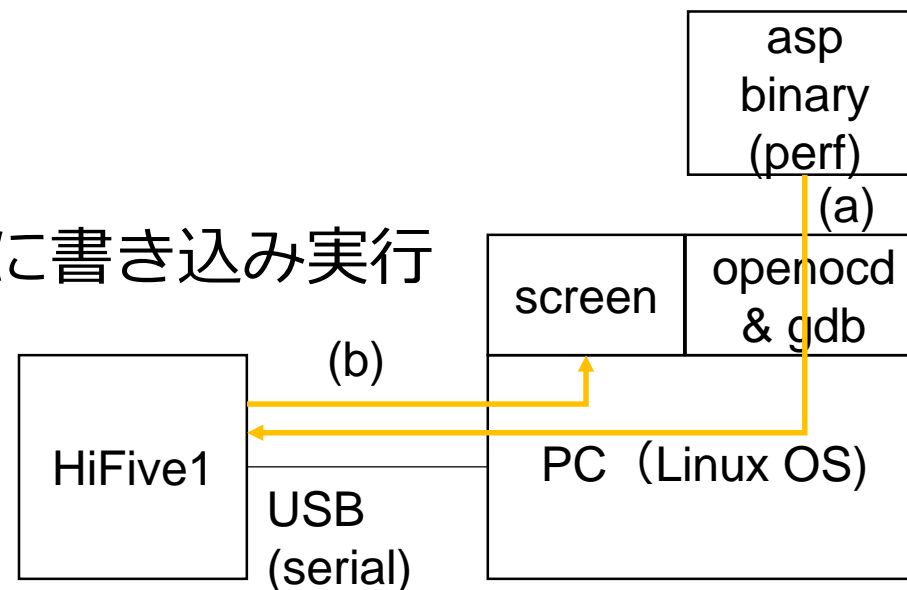
性能評価プログラムperfを実行して要所の遅延時間を計測

実行環境のセットアップ

- screenのインストール。
- openocdのインストール。openocd.cfgファイルを用意
- gdbのインストール。.gdbinitファイルを用意

実行環境

- (a) openocd & gdbにて、perfのbinaryをHiFive1に書き込み実行
 - text: 0x20000000(SPI Flash)
 - data: 0x80000000(DTIM)
- (b) screenでperfの実行結果を取得



TOPPERS/ASPに付属するperfの説明

perf0

- 何も実行しないでループしながら、時刻測定のオーバーヘッドを計測

perf1

- 低優先度タスクが動作中に高優先度タスクをwup_tskしてから実行開始するまでの時間計測
- 高優先度タスクがslp_tskしてから低優先度タスクが実行開始するまでの時間計測

perf4

- task switchを伴わないact_tskの処理時間の計測
- task switchを伴うact_tskの処理時間の計測
- 周期ハンドラから実行するiact_tsk（非タスクコンテキストからのact_tsk）の処理時間の計測

各perfは10000回の計測を繰り返し、実行完了後にヒストグラム形式でその結果を表示

問題点1：ビルド環境・実行環境の構築に手間がかかる

- toolchainのインストール（toolchainのビルド）
- 複数個所からソースやツールを取得
- aspビルドに必要なcfg（コンフィグレータ）が動作するのは32bit環境
 - Linux distributionによっては32bit supportがない
 - 32bitバイナリ動作のためにmulti arch設定が必要
- 計測のためにソースコード修正（後述）
- openocdの設定ファイルを用意
- gdbの設定ファイルを用意

⇒対策1：TOPPERS/ASPのビルド手順&実行手順をDocker化

問題点2：時刻測定関数(get_utm)の不具合

- perf0を走らせてみた結果。0か1000しか得られない

```
Performance evaluation program (0)
Measurement overhead
0 : 9988
> 1000 : 10
> INT_MAX : 2
```

- test_utm1を走らせると、デフォルトのget_utmに単調増加性がないことがわかる

```
system performance time goes back: 18616001(CYC) 18616000(TSK)
system performance time goes back: 18647001(TSK) 18647000(CYC)
system performance time goes back: 18649001(TSK) 18649000(CYC)
system performance time goes back: 18663001(TSK) 18663000(CYC)
system performance time goes back: 18667001(TSK) 18667000(CYC)
```

⇒ 対処2：マイクロ秒単位で計測するための時刻測定関数get_utmを別途用意

問題点3：RISC-V版のCPUの周波数が低い

- NUCLEO-F401RE版ASPは 84Mhzで動作
- 一方で、HiFive1版ASPは、デフォルトのビルドでCPUが16MHzで動作
- 周波数の差が激しい

⇒ **対策3：PLLの設定によりCPUの周波数を変えて計測**

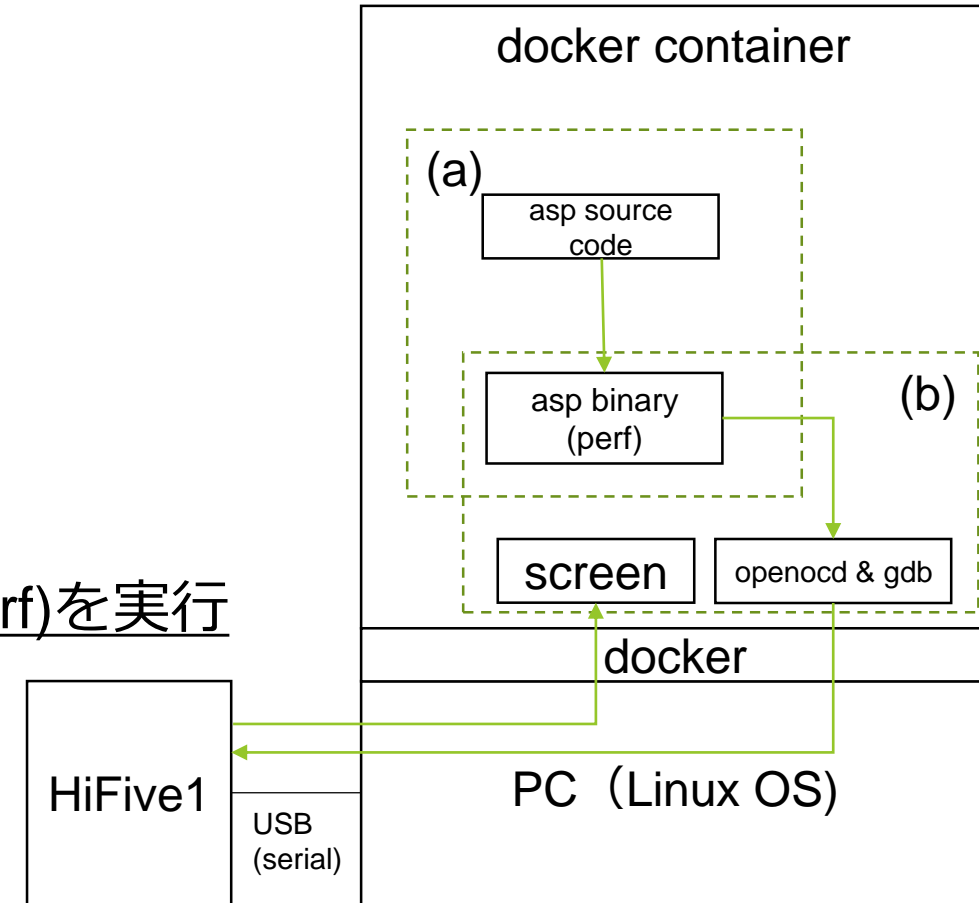
対処1： Docker化でビルド作業・実行作業を簡易化

(a) docker buildコマンドにて、aspのビルドを実行

- ビルド・実行に必要なパッケージをインストール
- asp source codeとコンフィギュレータの取得
- toolchain取得 & インストール
- 32bit用multi archの設定
- asp source codeにパッチ適用
- aspのビルド：各perfのバイナリ作成

(b) docker runコマンドにて、HiFive1ボード上でasp(perf)を実行

- docker内のscreen実行
- docker内のopenocd実行
- docker内のperfのバイナリを選択してgdb実行



対処2：マイクロ秒単位で計測するための関数get_utmを用意

CPUの周波数でカウントアップするレジスタを参照

- mcycle
- mcycleh

CPUの周波数は起動時に計測

- SystemFrequency

⇒レジスタ値からマイクロ秒に変換

```
+#ifdef OMIT_GET_UTM
+#include "target_timer.h"
+ER
+get_utm(SYSUTM *p_sysutm)
+{
+    SYSUTM utime = 0;
+    uint32_t mcycle_low, mcycle_high;
+    uint32_t mcycle_high_tmp;
+
+    do {
+        mcycle_high_tmp = read_csr(mcycleh);
+        mcycle_low = read_csr(mcycle);
+        mcycle_high = read_csr(mcycleh);
+    } while (mcycle_high_tmp != mcycle_high);
+
+    /* change clock count into micro sec */
+    utime += (SYSUTM)(mcycle_low / (SystemFrequency / 1000000));
+    utime += (SYSUTM)(UINT_MAX / (SystemFrequency / 1000000) * mcycle_high);
+
+    *p_sysutm = utime;
+
+    return(E_OK);
+}
+#endif
```

対処3：PLL設定でCPUの周波数調整

- CPUの周波数は、aspデフォルトのビルドで16MHz
- ビルド時にDEFAULT_CLOCKマクロをONにする
 - PLLの仕様から、計算上は256Mhzになる
 - ただし、実測してみると 280Mhz 以上出ている
- PLL設定で周波数を変更
 - PLLの仕様から計算では、 $R=1$ $F=41$ $Q=3$ とすれば84Mhzになるが、実測では合致しなかった
 - $R=1$ $F=37$ $Q=3$ で実測値が約86Mhzとなったので、この設定で計測

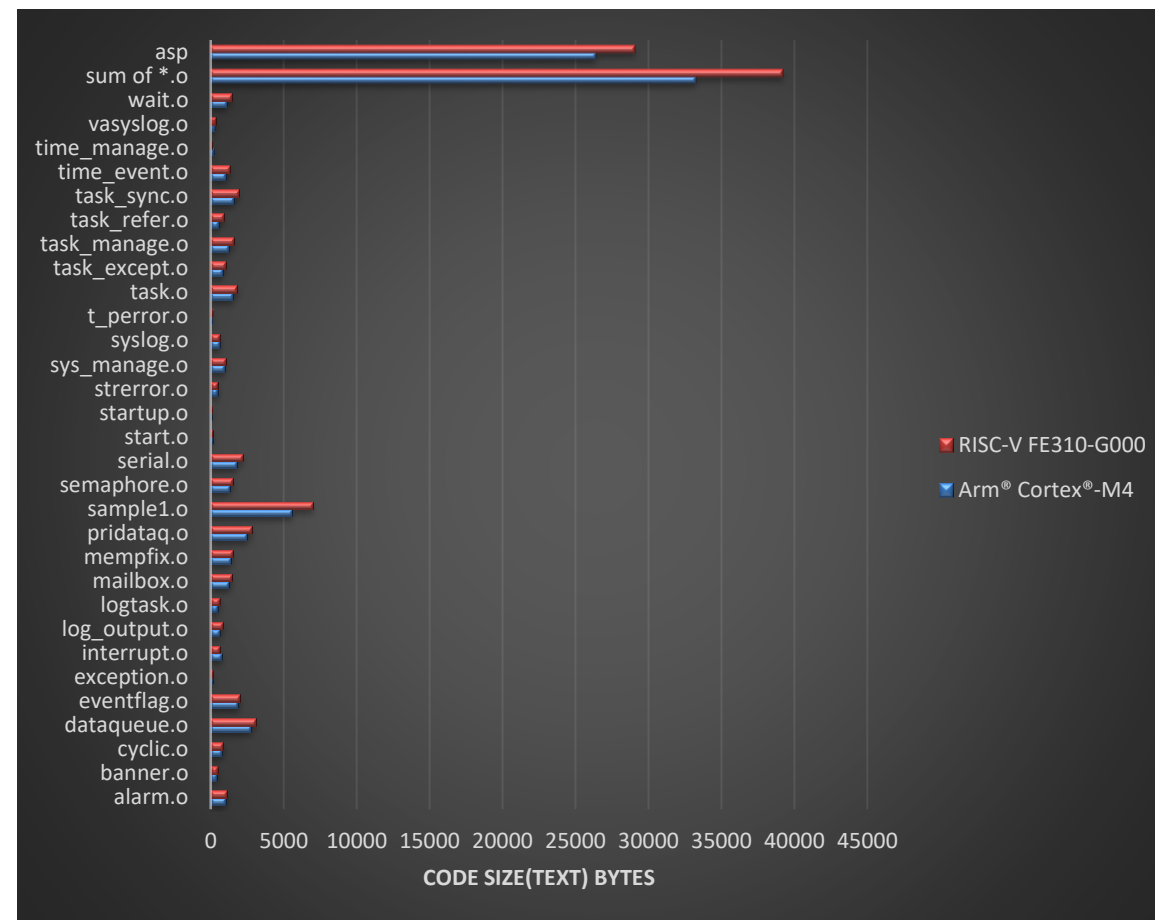
コードサイズの評価

使用したコンパイラ（両方とも圧縮命令あり）

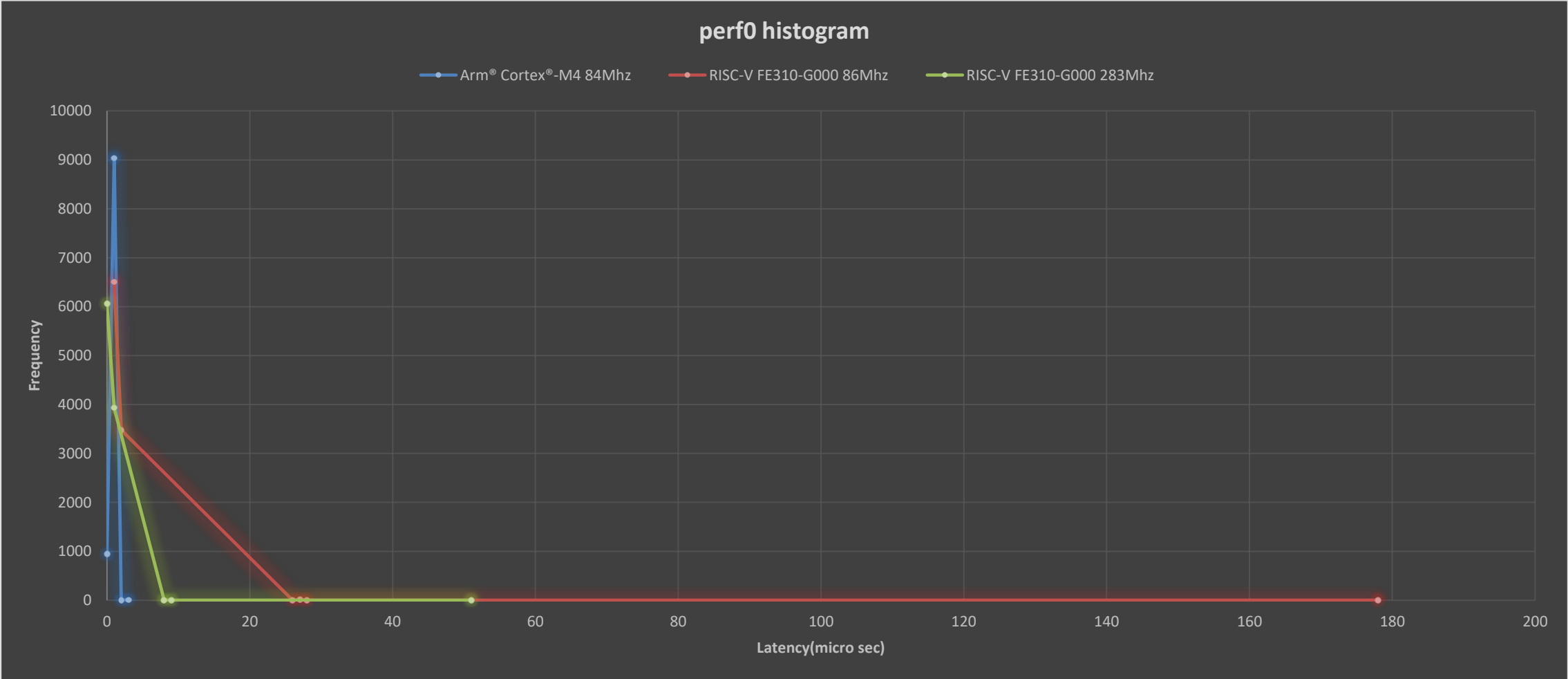
- Arm® core : gcc version 8.3.1 20190703 (release) [gcc-8-branch revision 273027] (GNU Tools for Arm Embedded Processors 8-2019-q3-update)
- RISC-V : gcc version 8.3.0 (SiFive GCC 8.3.0-2019.08.0)

コードサイズ(text領域)の比較

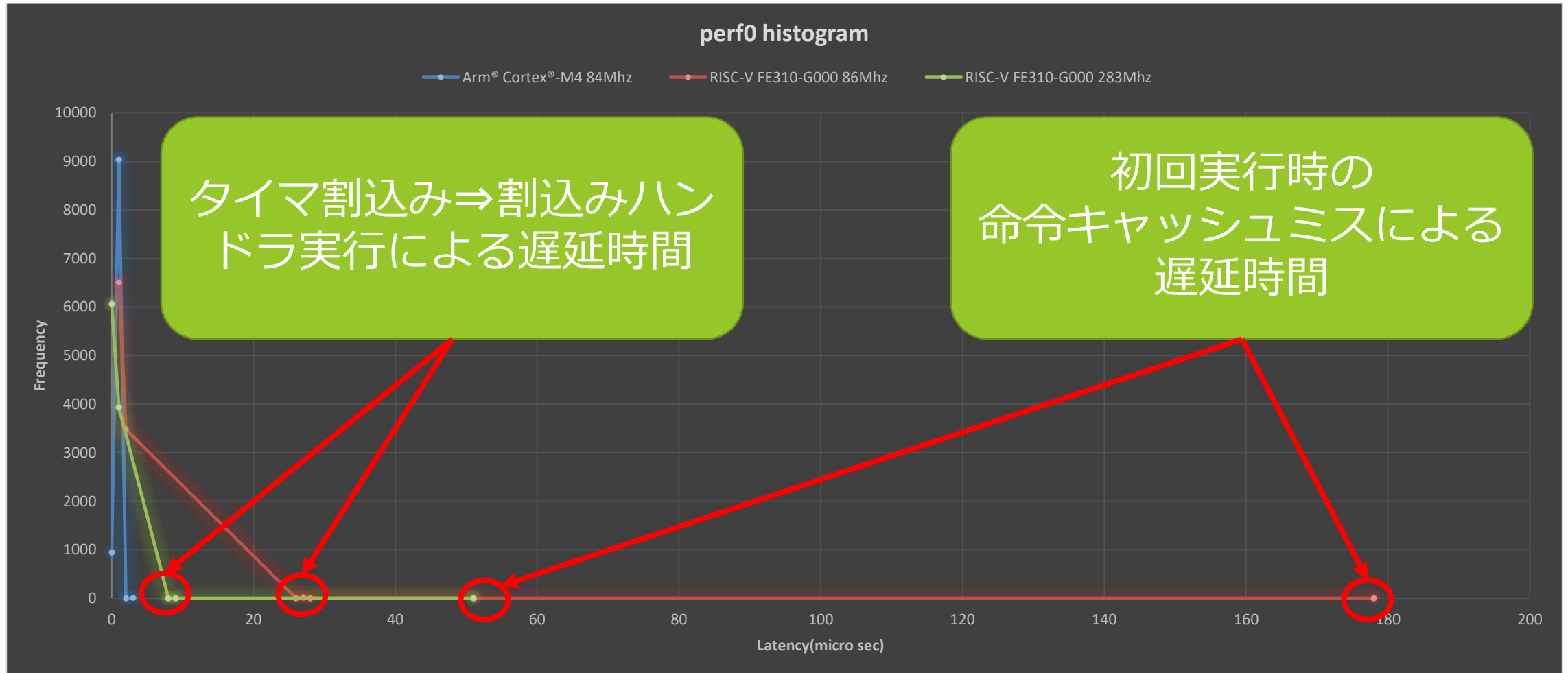
- 共通のobject fileの総和では、RISC-VがArm® coreよりも17.8%増加
- asp単体での比較は、RISC-VがArm® coreよりも10.3%増加



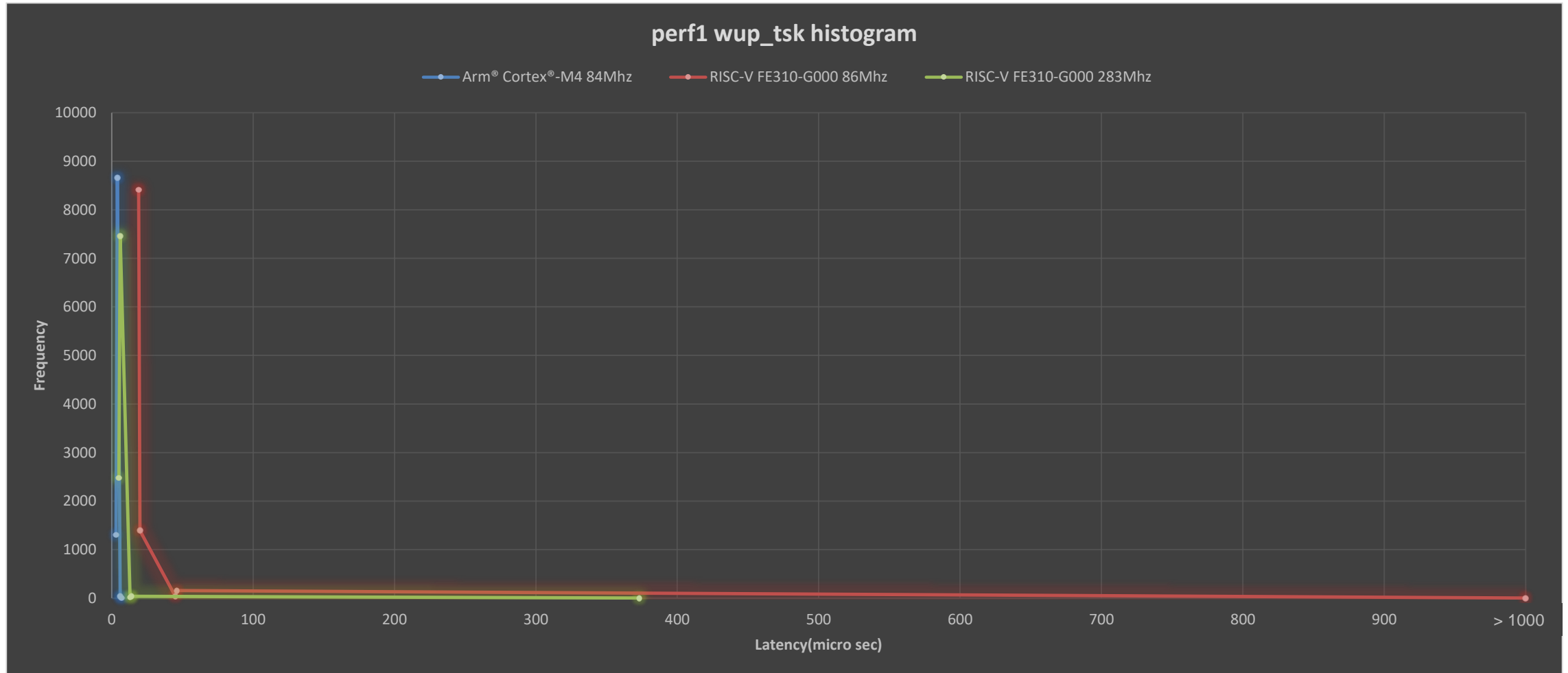
perf0 : 測定オーバヘッドの評価



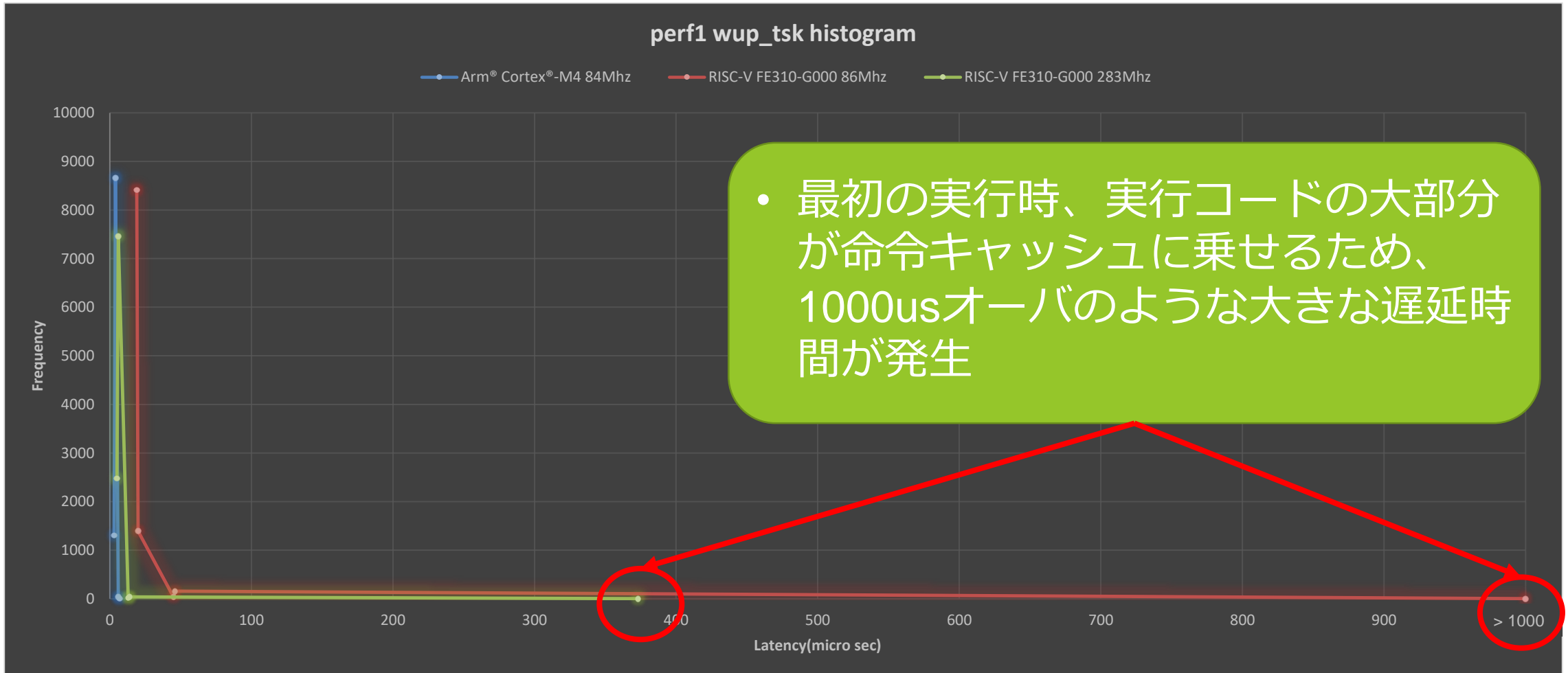
perf0 : 測定オーバーヘッドの評価 : 遅延要因の考察



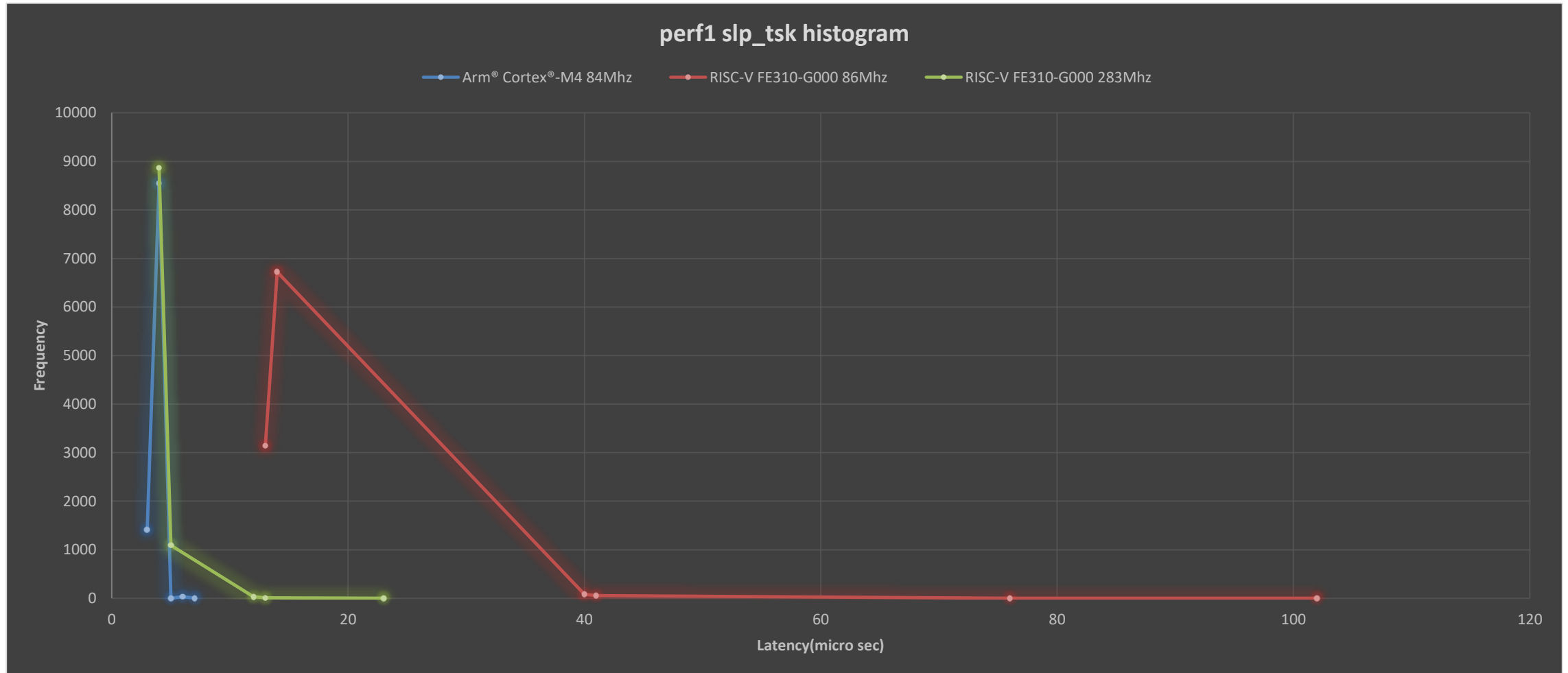
perf1 : wup_tskによるタスク切り替え時間の評価



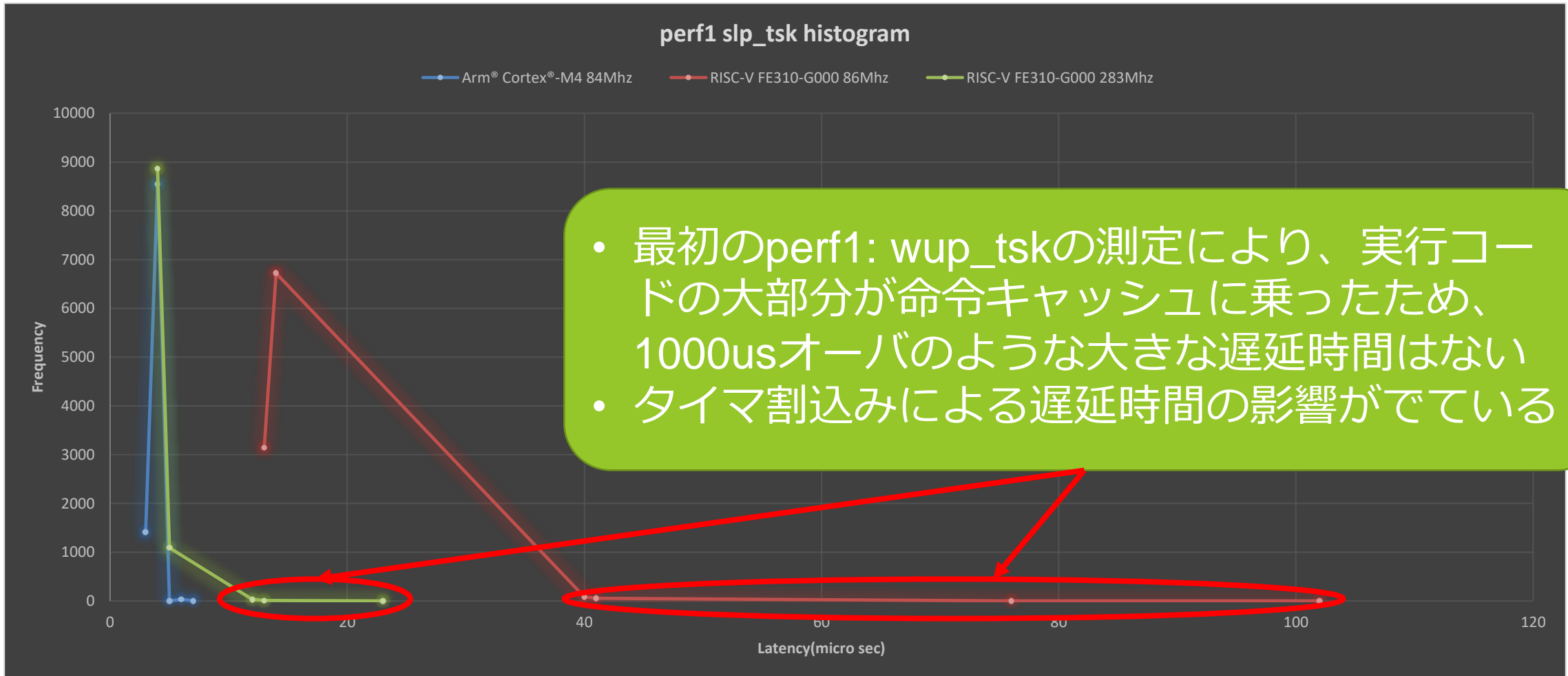
perf1 : wup_tskによるタスク切り替え時間の評価：遅延要因の考察



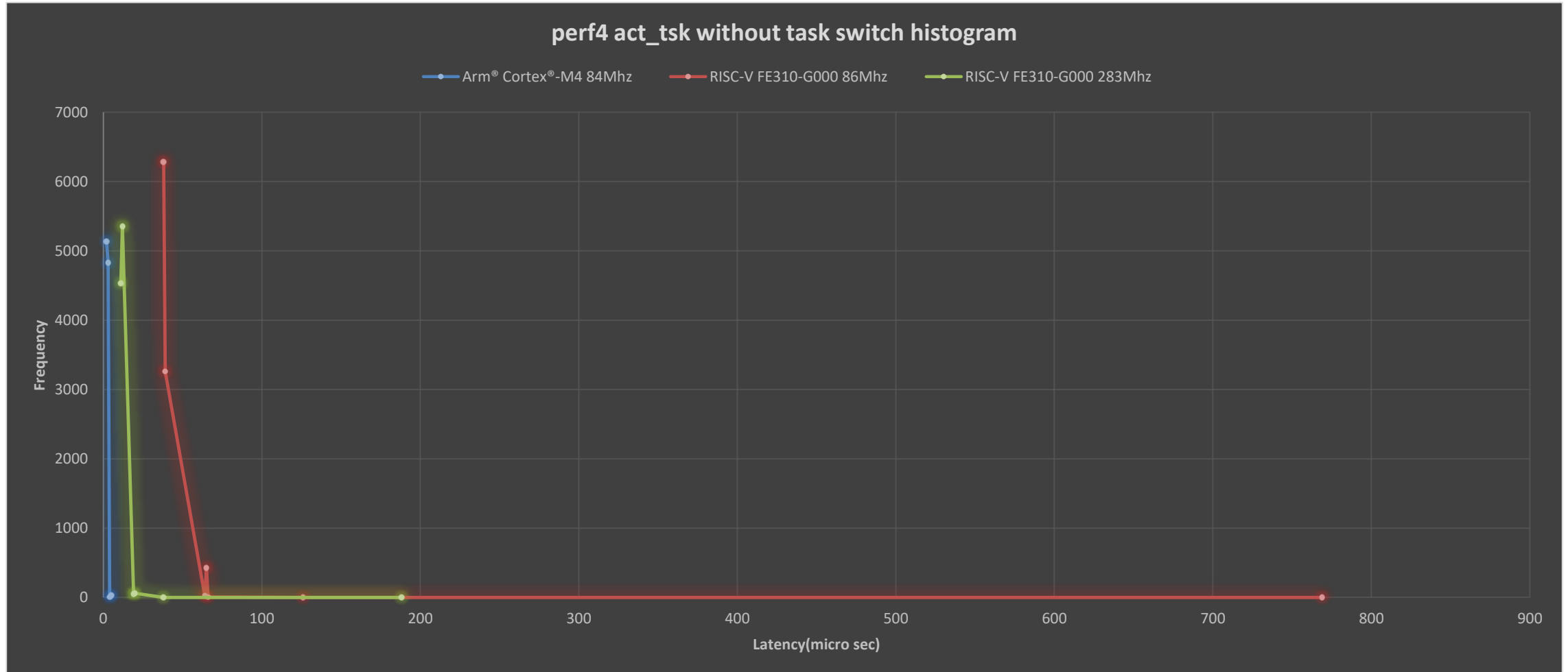
perf1 : slp_tskによるタスク切り替え時間の評価



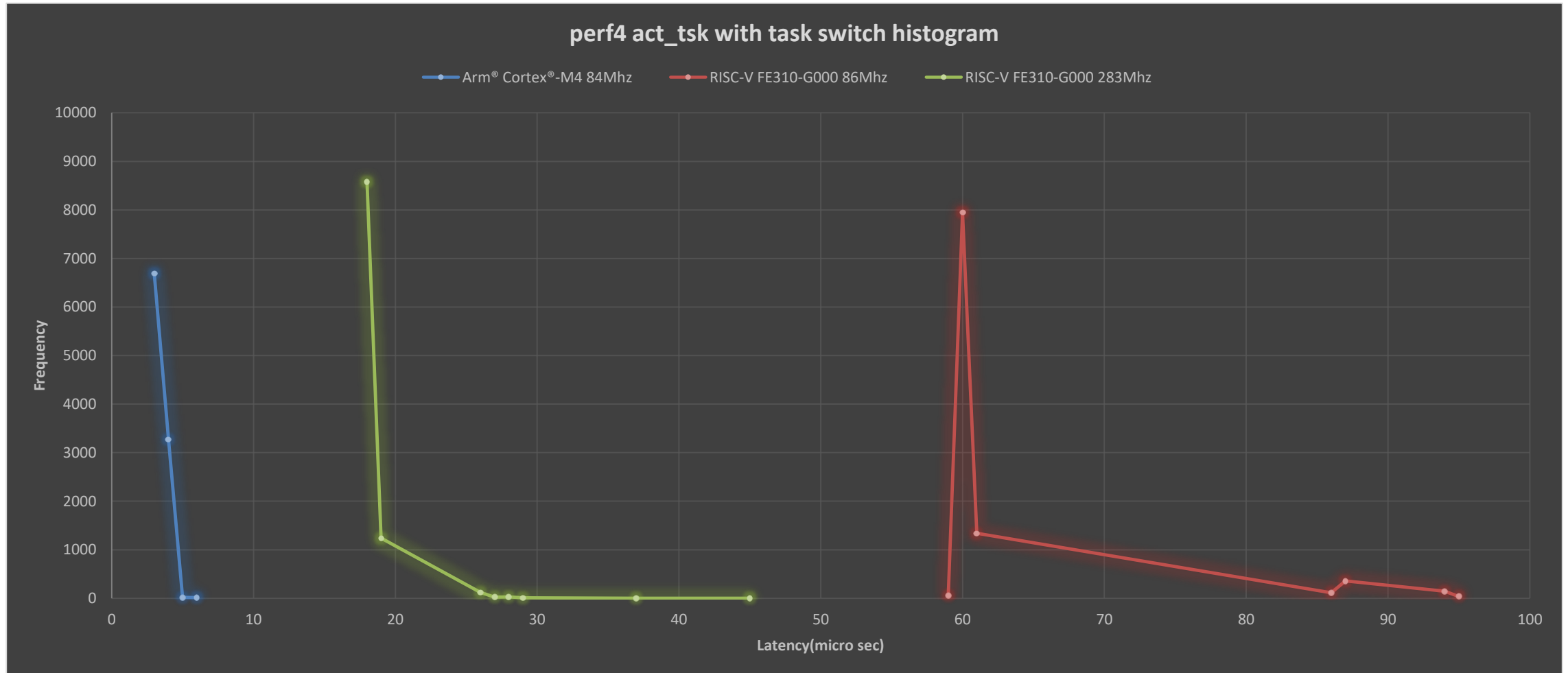
perf1 : slp_tskによるタスク切り替え時間の評価：遅延要因の考察



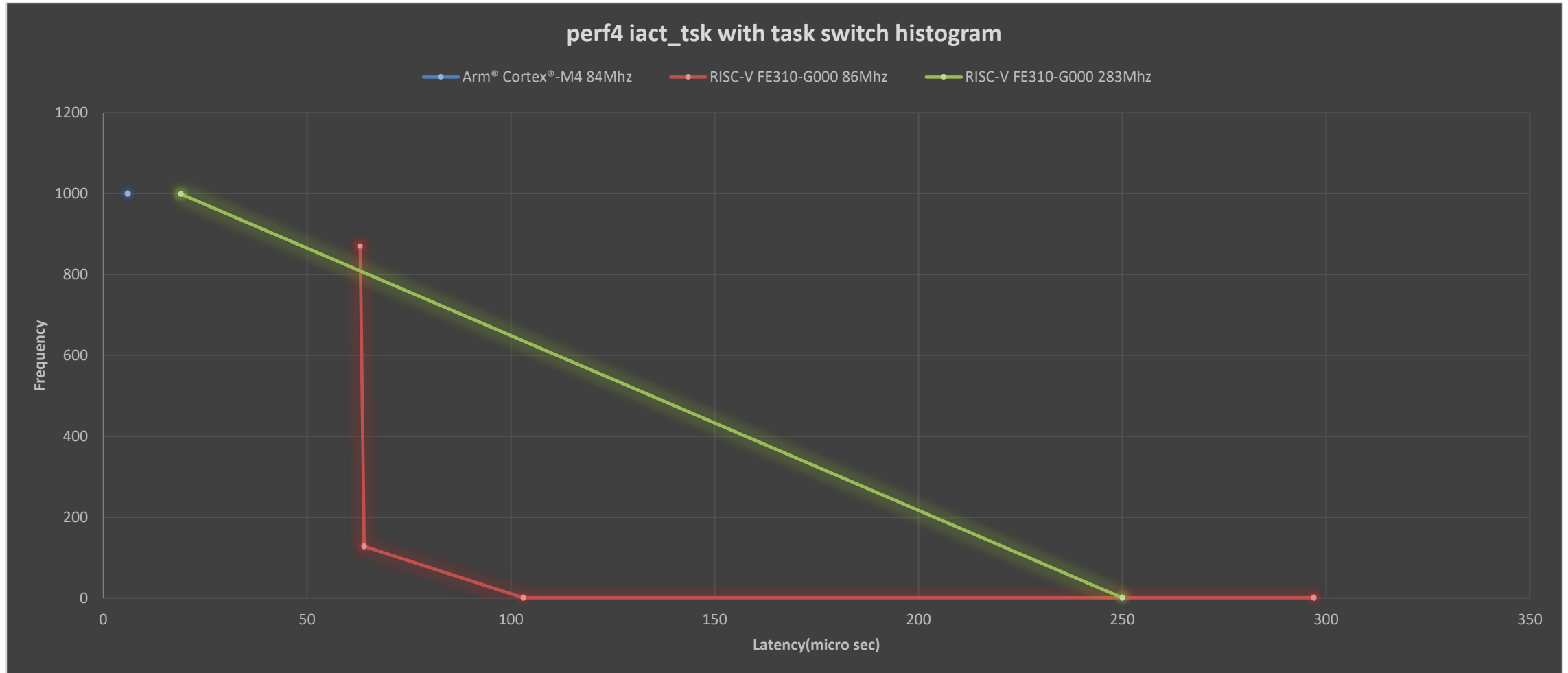
perf4 : task switchを伴わないact_tskの実行時間の評価



perf4 : task switchを伴うact_tskの実行時間の評価



perf4 : iact_tskの実行時間の評価



TOPPERS/ASP on HiFive1を使用してコードサイズとperfの計測を行った

- 計測する際に生じた課題は以下のように対処
 - ビルド環境・実行環境の構築の手間⇒dockerによる簡略化
 - 時刻測定の不具合⇒測定用にget_utmを用意
 - CPU周波数⇒PLLにより動作周波数の変更
- 評価結果：コードサイズ
 - Arm® Cortex®-M4版ASPの各object fileの総和と比較して、おおよそ17.8%ほど増加
- 評価結果：perfによる遅延時間の測定
 - 命令キャッシュの影響とタイマ割込みによる遅延時間が目立つ
 - タイマ割込みは、Arm® coreのsysticのように自動でreloadするタイプなら低減できそう
 - 命令キャッシュミスは、起動時にFW等で命令キャッシュに命令コードをloadする仕組み or キャッシュにコードを乗せるための空運転が必要

KIOXIA

Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.
Other company names, product names, and service names may be trademarks of their respective companies.